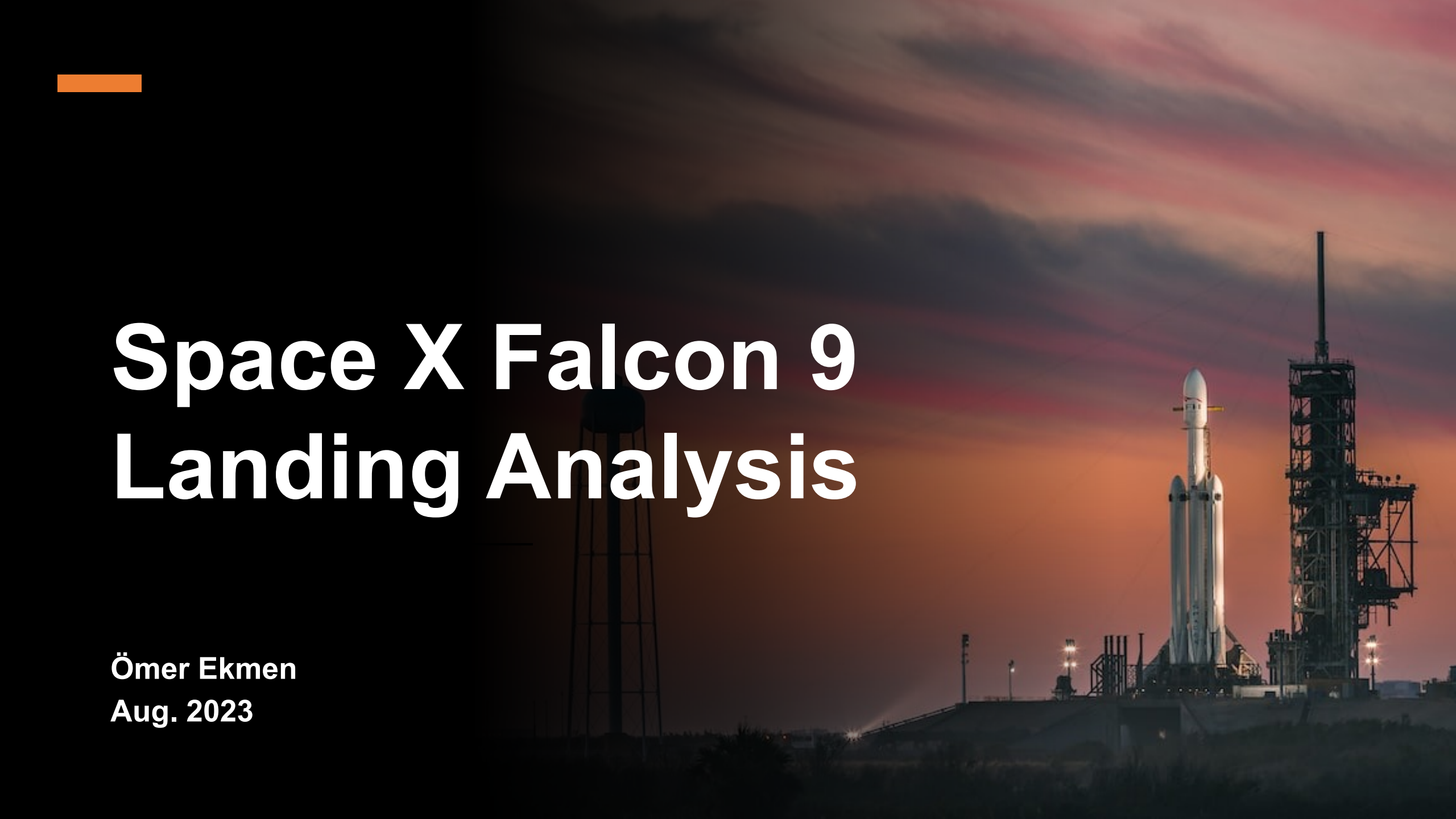


Space X Falcon 9 Landing Analysis

Ömer Ekmen

Aug. 2023

Outline

Executive Summary

Summary of Methodologies:

This project follows these steps:

- Data Collection
- Data Wrangling
- Exploratory Data Analysis
- Interactive Visual Analytics
- Predictive Analysis (Classification)

Summary of Results:

This project produced the following outputs and visualizations:

1. Exploratory Data Analysis (EDA) results
2. Geospatial analytics
3. Interactive dashboard
4. Predictive analysis of classification models

Introduction

- SpaceX has emerged as a pathfinder, revolutionizing the field of rocket technology and launch economics in the enormous field of space exploration, where every gram of payload and every dollar of investment matter. The Falcon 9 rocket, a symbol of innovation and reusability that has changed the dynamics of reasonably priced space travel, is at the center of this transition. The Falcon 9's capability to land and then reuse its first stage is evidence of SpaceX's determination to completely transform the space industry.
- Within the larger field of predictive analytics and aerospace engineering, the ***SpaceX Falcon 9 First Stage Landing Analysis Project*** is a significant undertaking. This project's main objective is to use data science to predict how the Falcon 9 first stage landing will turn out. We can evaluate the cost-effectiveness of each launch and offer crucial information for decision-making in the space sector by correctly forecasting whether the first stage will successfully return and land.

METHODOLOGY

Methodology

1. Data Collection

The project begins with the collection of pertinent data by making ***GET*** requests to the ***SpaceX REST API*** and employing web scraping techniques. This step is pivotal in amassing a substantial dataset that encapsulates the essential variables related to Falcon 9 launches and landings.

2. Data Wrangling

Data wrangling is a crucial step in the SpaceX Falcon 9 Landing Analysis project. It involves:

- **Handling Missing Values**: The ***.fillna()*** method replaces missing values with appropriate placeholders, ensuring data consistency.
- **Insights with *.value_counts():*** This method helps us understand:
  - Number of launches at each site.
  - Occurrence of each orbit type.
  - Occurrence of mission outcomes for each orbit type.
- **Creating Landing Outcome Label:** A binary label is created to signify successful (1) or unsuccessful (0) booster landings. This label is pivotal for training classification models.

3. Exploratory Data Analysis

The project delves deeper into the data through exploratory data analysis, leveraging ***SQL*** queries for advanced data manipulation. Employing ***Pandas*** and ***Matplotlib***, the project constructs visualizations to illuminate relationships between variables and identify patterns within the dataset.

4. Interactive Visual Analytics

Taking the analysis further, the project incorporates geospatial analytics using ***Folium***, enabling the visualization of geographical aspects of launches. Additionally, the creation of an interactive dashboard using ***Plotly Dash*** enhances data representation and engagement, fostering a dynamic exploration of the dataset.

5. Data Modelling and Evaluation

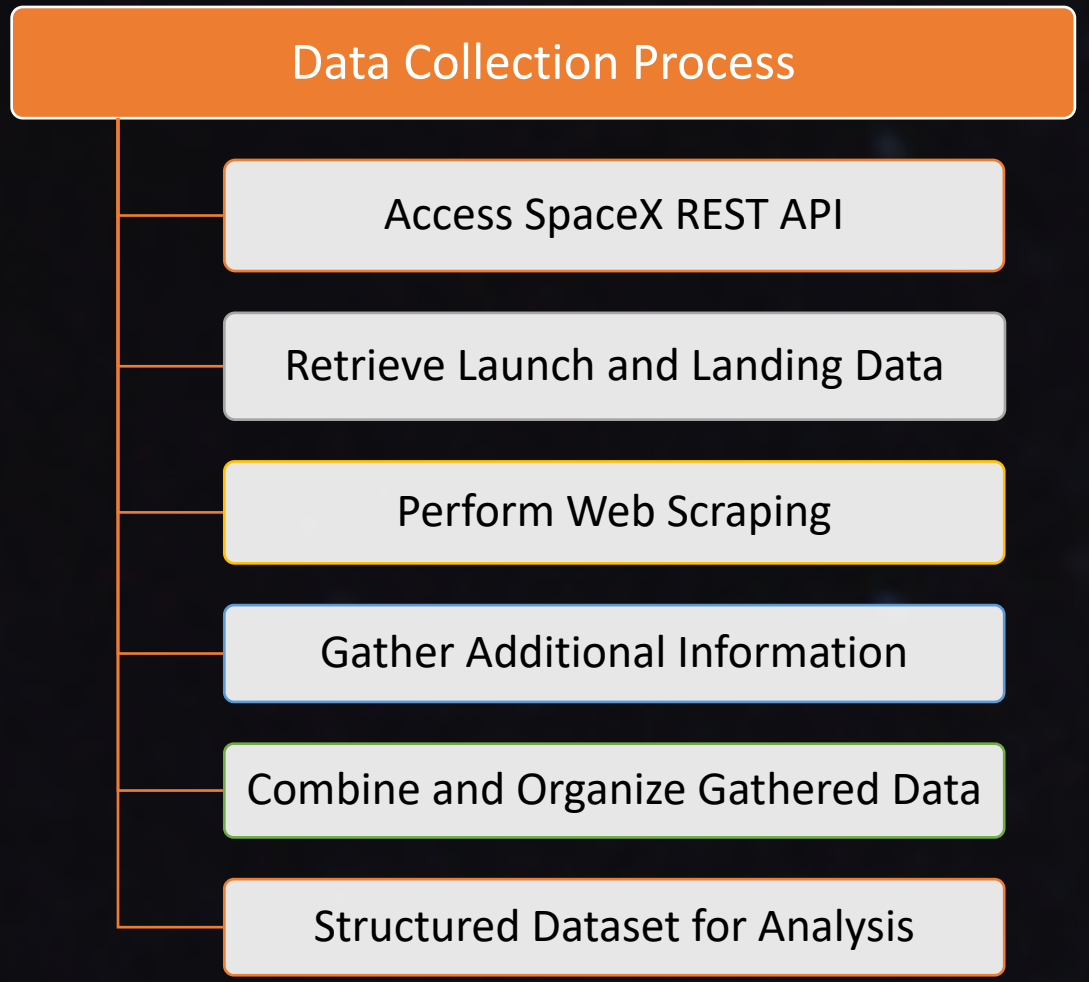
The core of the project revolves around data modeling and evaluation. Scikit-Learn comes into play for pre-processing tasks like standardizing data, followed by the division of the dataset into training and testing subsets using ***train_test_split***. Multiple classification models are trained on the data, allowing for the prediction of landing outcomes. The use of ***GridSearchCV*** helps identify optimal hyperparameters, enhancing the models' performance. The project culminates in the creation of confusion matrices and the assessment of classification model accuracy, providing a clear understanding of the model's predictive power.

Data Collection

Accessing SpaceX REST API: The SpaceX REST API is queried to retrieve detailed information about Falcon 9 launches and landing outcomes. This includes launch site, orbit, mission success, and landing outcome data.

Web Scraping for Supplementary Data: In cases where certain data isn't available through the API, web scraping techniques are employed to gather additional information, such as launch site details, mission objectives, and other relevant attributes.

Combining and Structuring Data: The collected data is organized into a structured dataset that captures key variables required for the analysis, such as launch site, orbit, and mission outcome.



Data Collection – SpaceX API

Obtaining launch-related data from the SpaceX API, such as details on the rocket launched, the payload delivered, launch specs, landing specifications, and landing outcome.

1

- Make a GET response to the SpaceX REST API
- Convert the response to a .json file then to a Pandas DataFrame

```
spacex_url="https://api.spacexdata.com/v4/launches/past"

response = requests.get(spacex_url)
```

```
# Use json_normalize meethod to convert the json result into a dataframe
data = pd.json_normalize(response.json())
```

2

- Use custom logic to clean the data
- Define lists for data to be stored in
- Call custom functions to retrieve data and fill the lists
- Use these lists as values in a dictionary and construct the dataset

```
#Global variables
BoosterVersion = []
PayloadMass = []
Orbit = []
LaunchSite = []
Outcome = []
Flights = []
GridFins = []
Reused = []
Legs = []
LandingPad = []
Block = []
ReusedCount = []
Serial = []
Longitude = []
Latitude = []
```

```
# Call getBoosterVersion
getBoosterVersion(data)

# Call getLaunchSite
getLaunchSite(data)

# Call getPayloadData
getPayloadData(data)

# Call getCoreData
getCoreData(data)
```

```
launch_dict = {'FlightNumber': list(data['flight_number']),
'Date': list(data['date']),
'BoosterVersion':BoosterVersion,
'PayloadMass':PayloadMass,
'Orbit':Orbit,
'LaunchSite':LaunchSite,
'Outcome':Outcome,
'Flights':Flights,
'GridFins':GridFins,
'Reused':Reused,
'Legs':Legs,
'LandingPad':LandingPad,
'Block':Block,
'ReusedCount':ReusedCount,
'Serial':Serial,
'Longitude': Longitude,
'Latitude': Latitude}
```

3

- Create a Pandas DataFrame from the constructed dictionary dataset

```
# Create a data from launch_dict
df = pd.DataFrame(launch_dict)
```

4

- Filter the DataFrame to only include Falcon 9 launches
- Reset the FlightNumber column
- Replace missing values of PayloadMass with the mean PayloadMass value

```
data_falcon9 = df[df['BoosterVersion'] == 'Falcon 9']
```

```
data_falcon9.loc[:,'FlightNumber'] = list(range(1, data_falcon9.shape[0]+1))
```

```
# Calculate the mean value of PayloadMass column
mean = data_falcon9['PayloadMass'].mean()
# Replace the np.nan values with its mean value
data_falcon9['PayloadMass'].replace(np.nan, mean, inplace=True)
```

Github Link

Data Collection – Web Scraping

Our approach involves employing web scraping techniques to extract comprehensive historical launch records of Falcon 9. Specifically, we target the relevant information from the Wikipedia page titled "List of Falcon 9 and Falcon Heavy launches." Through this process, we aim to accumulate a thorough dataset detailing the launch history of Falcon 9 rockets.

1
- Request the HTML page from the static URL
- Assign the response to an object

```
static_url = "https://en.wikipedia.org/w/index.php?title=List_of_Falcon_9_and_Falcon_Heavy_launches&oldid=1027686922"
```

```
# use requests.get() method with the provided static_url
# assign the response to a object
request = requests.get(static_url).text
```

2
- Create a BeautifulSoup object from the HTML response object
- Find all tables within the HTML page

```
# Use BeautifulSoup() to create a BeautifulSoup object from a response text content
soup = BeautifulSoup(request, 'html.parser')
```

```
# Use the find_all function in the BeautifulSoup object, with element type `table`
# Assign the result to a list called `html_tables`
html_tables = soup.find_all('table')
```

3
- Collect all column header names from the tables found within the HTML page

```
column_names = []
for name in first_launch_table.find_all('th'):
    column_names.append(extract_column_from_header(name))

print("Column names: ", column_names, "\n")
```

4
- Use the column names as keys in a dictionary
- Use custom functions and logic to parse all launch tables to fill the dictionary values

```
launch_dict= dict.fromkeys(column_names)

# Remove an irrelvant column
del launch_dict['Date and time ( )']

# Let's initial the launch_dict with each value to be an empty list
launch_dict['Flight No.'] = []
launch_dict['Launch site'] = []
launch_dict['Payload'] = []
launch_dict['Payload mass'] = []
launch_dict['Orbit'] = []
launch_dict['Customer'] = []
launch_dict['Launch outcome'] = []
# Added some new columns
launch_dict['Version Booster']=[]
launch_dict['Booster landing']=[]
launch_dict['Date']=[]
launch_dict['Time']=[]
```

5
- Convert the dictionary to a Pandas DataFrame ready for export

```
df=pd.DataFrame(launch_dict)
```

Github Link

Data Wrangling

- The SpaceX dataset incorporates diverse launch facilities, with each specific location being represented in the "**LaunchSite**" column. Meanwhile, individual launches are tailored to achieve specific orbits, with various well-defined orbit types exemplified in the figure below. These orbit types are captured in the "**Orbit**" column of the dataset.
- The landing outcome is shown in the **Outcome** column:
  - **True Ocean** – the mission outcome was successfully landed to a specific region of the ocean
  - **False Ocean** – the mission outcome was unsuccessfully landed to a specific region of the ocean.
  - **True RTLS** – the mission outcome was successfully landed to a ground pad
  - **False RTLS** – the mission outcome was unsuccessfully landed to a ground pad.
  - **True ASDS** – the mission outcome was successfully landed to a drone ship
  - **False ASDS** – the mission outcome was unsuccessfully landed to a drone ship.
  - **None ASDS** and **None None** – these represent a failure to land.

Github Link

Data Wrangling

Initial Data Exploration
Using the .value_counts() method to determine the following:

1. Number of launches on each site

```
# Apply value_counts() on column LaunchSite
df['LaunchSite'].value_counts()
```

```
CCAFS SLC 40    55
KSC LC 39A      22
VAFB SLC 4E     13
Name: LaunchSite, dtype: int64
```

2. Number and occurrence of each orbit

```
# Apply value_counts on Orbit column
df['Orbit'].value_counts()
```

```
GTO      27
ISS      21
VLEO     14
PO        9
LEO       7
SSO       5
MEO       3
GEO       1
HEO       1
ES-L1     1
SO        1
Name: Orbit, dtype: int64
```

Github Link

3. Number and occurrence of landing outcome per orbit type

```
# landing_outcomes = values on Outcome column
landing_outcomes = df['Outcome'].value_counts()
```

```
True ASDS      41
None None      19
True RTLS      14
False ASDS      6
True Ocean      5
None ASDS       2
False Ocean     2
False RTLS      1
Name: Outcome, dtype: int64
```

Data Wrangling
In order to ascertain the success of booster landings, a binary column is optimal. This entails utilizing a column with values of either 1 or 0, signifying the landing's success. This procedure involves:

4. Defining a set of unsuccessful (bad) outcomes, `bad_outcome`

```
bad_outcomes=set(landing_outcomes.keys()[[1,3,5,6,7]])
bad_outcomes
```

```
{'False ASDS', 'False Ocean', 'False RTLS', 'None ASDS', 'None None'}
```

5. Creating a list, **landing_class**, where the element is 0 if the corresponding row in **Outcome** is in the set **bad_outcome**, otherwise, it's 1.

```
landing_class = []
for outcome in df['Outcome']:
    if outcome in bad_outcomes:
        landing_class.append(0)
    else:
        landing_class.append(1)
```

6. Create a **Class** column that contains the values from the list **landing_class**

```
df['Class']=landing_class
df[['Class']].head(8)
```

7. Export the DataFrame as a .csv file.

```
df.to_csv("dataset_part_2.csv", index=False)
```

EDA with Data Visualization

SCATTER CHARTS

Scatter charts were produced to visualize the relationships between:

- Flight Number and Launch Site
- Payload and Launch Site
- Orbit Type and Flight Number
- Payload and Orbit Type

When examining links or correlations between two numerical variables, scatter charts are helpful.

BAR CHART

A bar chart was produced to visualize the relationship between:

- Success Rate and Orbit Type

To compare a numerical value to a categorical variable, we utilize bar charts. Depending on how much data there is, bar charts can be horizontal or vertical.

LINE CHART

Line charts were produced to visualize the relationships between:

- Success Rate and Year (i.e. the launch success yearly trend)

Line charts, which often demonstrate how a variable changes over time, have numerical values on both axes.

Github Link

EDA with SQL

SQL queries were run in order to glean some data about the dataset.

The data set was subjected to SQL queries that were used to:

- Show the names of the distinct space mission launch sites.
- Show 5 records where the string begin with 'CCA' in the launch sites.
- Show the total payload mass carried by all NASA (CRS) launched rockets.
- Display the booster version F9 v1.1's average payload mass.
- Indicate the day that the first successful ground landing took place.
- Name the rockets that were successful with a drone ship and a payload weighing between 4,000 and 6,000 kg.
- List the total number of successful and failed mission outcomes
- List the names of the booster versions which have carried the maximum payload mass
- List the failed landing outcomes on drone ships, their booster versions, and launch site names for 2015
- Rank the count of landing outcomes (such as Failure (drone ship) or Success (ground pad)) between the date 2010-06-04 and 2017-03-20, in descending order

Github Link

Interactive Map with Folium

In the SpaceX Falcon 9 Landing Analysis project, a dynamic and informative interactive map has been constructed using the Folium library. Various map objects, including markers, circles, and lines, have been strategically integrated to enhance the visualization of key data points. Here's a summary of the map objects and their significance:

- **Markers:** Launch sites are pinpointed with markers, clarifying launch origins.
- **Circles:** Successful landing zones are marked by circles, representing landing coordinates and radius.
- **Lines:** Launch trajectories are visualized through lines connecting launch to landing sites.

1. Mark all launch sites on a map
   - Initialise the map using a **Folium Map** object
   - Add a **folium.Circle** and **folium.Marker** for each launch site on the launch map
2. Mark the success/failed launches for each site on a map
   - As many launches have the same coordinates, it makes sense to cluster them together.
   - Before clustering them, assign a marker colour of successful (class = 1) as green, and failed (class = 0) as red.
   - To put the launches into clusters, for each launch, add a **folium.Marker** to the **MarkerCluster()** object.
   - Create an icon as a text label, assigning the **icon_color** as the **marker_color** determined previously.
3. Calculate the distances between a launch site to its proximities
   - To explore the proximities of launch sites, calculations of distances between points can be made using the Lat and Long values.
   - After marking a point using the Lat and Long values, create a **folium.Marker** object to show the distance.
   - To display the distance line between two points, draw a **folium.PolyLine** and add this to the map.

Github Link

Dashboard with Plotly Dash

- In the SpaceX Falcon 9 Landing Analysis, a Plotly Dash dashboard has been meticulously crafted, featuring a collection of interactive plots and graphs. These visual elements are thoughtfully designed to present critical insights and enable users to interactively explore the dataset. Here's a summary of the dashboard components and their significance:

1. Pie chart (**px.pie()**) showing the total successful launches per site
   - This makes it clear to see which sites are most successful
   - The chart could also be filtered (using a **dcc.Dropdown()** object) to see the success/failure ratio for an individual site
2. Scatter graph (**px.scatter()**) to show the correlation between outcome (success or not) and payload mass (kg)
   - This could be filtered (using a **RangeSlider()** object) by ranges of payload masses
   - It could also be filtered by booster version

Github Link

Predictive Analysis (Classification)

In the SpaceX Falcon 9 Landing Analysis, the development of a classification model to predict landing outcomes is a comprehensive process involving iterative steps for building, evaluating, and improving model performance. Here's a summarized overview of the model development process along with key phrases and a simplified flowchart:

1. **Data Preprocessing:** The dataset is preprocessed, including standardization and splitting into training and testing subsets.
2. **Model Building:** Various classification algorithms, such as Logistic Regression, Random Forest, and Support Vector Machines, are employed to build initial models.

   To prepare the dataset for model development:

   - Load dataset
   - Perform necessary data transformations (standardise and pre-process)
   - Split data into training and test data sets, using **train_test_split()**
   - Decide which type of machine learning algorithms are most appropriate

   For each chosen algorithm:

   - Create a **GridSearchCV** object and a dictionary of parameters
   - Fit the object to the parameters
   - Use the training data set to train the model

3. **Model Evaluation:** For each chosen algorithm:

   Using the output **GridSearchCV** object:

   - Check the tuned hyperparameters (**best_params_**)
   - Check the accuracy (**score** and **best_score_**)

   Plot and examine the Confusion Matrix

4. **Model Selection:** The best-performing model is selected based on evaluation metrics.
   - Review the accuracy scores for all chosen algorithms

Github Link

Results

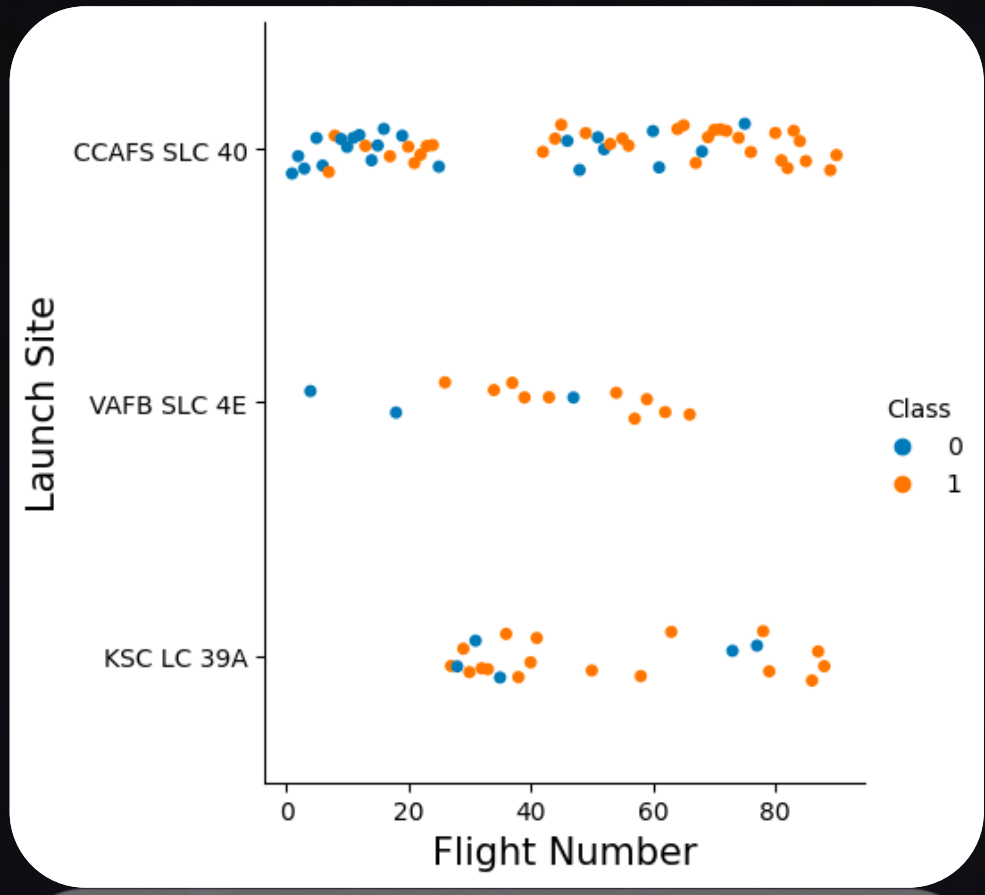
- EDA Results
- Interactive Analytics Results
- Predictive Analysis Results

EDA RESULTS – WITH VISUALIZATION

Flight Number vs. Launch Site

The scatter plot of Launch Site vs. Payload Mass shows that:

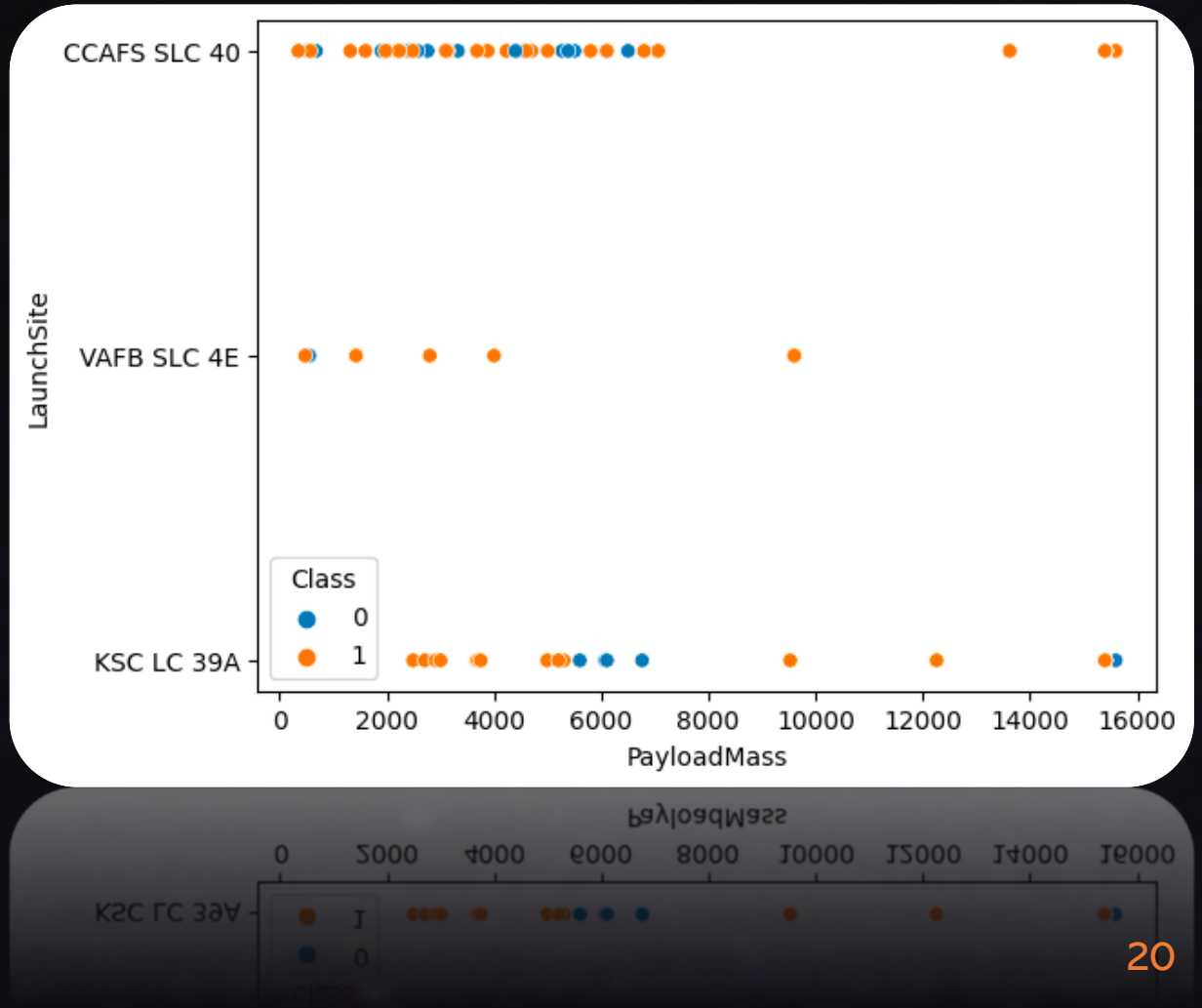
- Above a payload mass of around 7000 kg, there are very few unsuccessful landings, but there is also far less data for these heavier launches.
- There is no clear correlation between payload mass and success rate for a given launch site.
- All sites launched a variety of payload masses, with most of the launches from CCAFS SLC 40 being comparatively lighter payloads (with some outliers).



Payload Mass vs. Launch Site

The scatter plot of Launch Site vs. Payload Mass shows that:

- Above a payload mass of around 7000 kg, there are very few unsuccessful landings, but there is also far less data for these heavier launches.
- There is no clear correlation between payload mass and success rate for a given launch site.
- All sites launched a variety of payload masses, with most of the launches from CCAFS SLC 40 being comparatively lighter payloads (with some outliers).



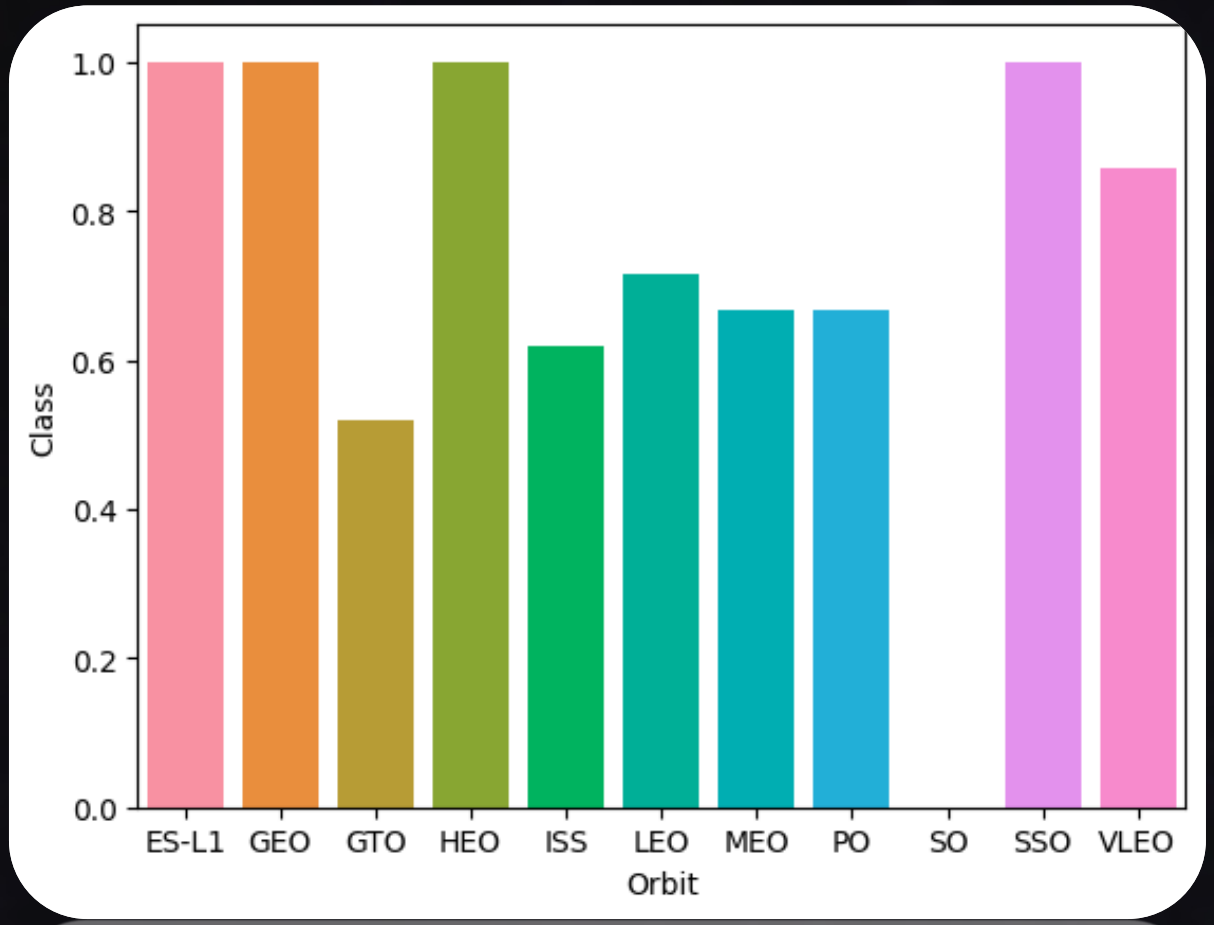
Success Rate vs. Orbit Type

The bar chart of Success Rate vs. Orbit Type shows that the following orbits have the highest (100%) success rate:

- ES-L1 (Earth-Sun First Lagrangian Point)
- GEO (Geostationary Orbit)
- HEO (High Earth Orbit)
- SSO (Sun-synchronous Orbit)

The orbit with the lowest (0%) success rate is:

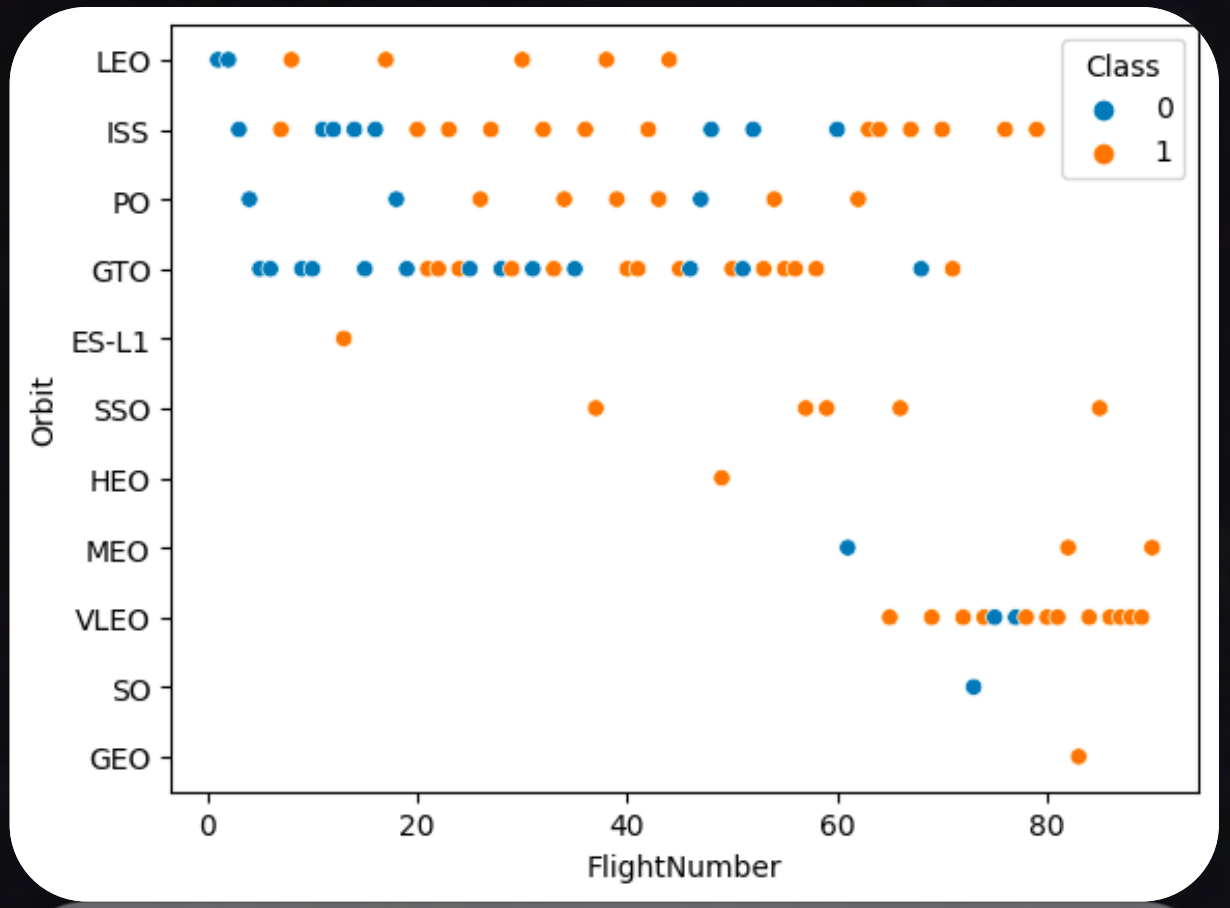
- SO (Heliocentric Orbit)



Flight Number vs. Orbit Type

This scatter plot of Orbit Type vs. Flight number shows a few useful things that the previous plots did not, such as:

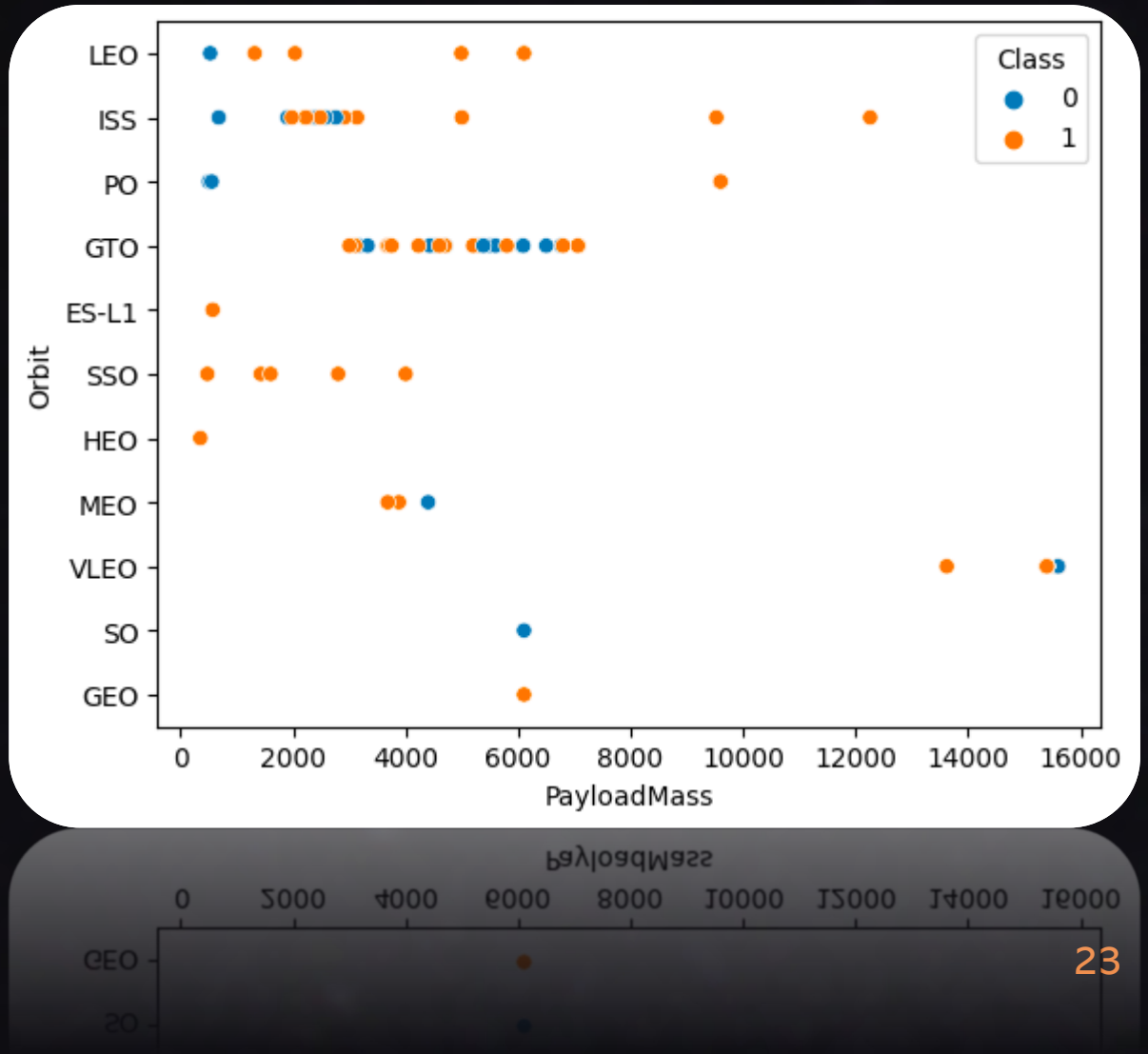
- The 100% success rate of GEO, HEO, and ES-L1 orbits can be explained by only having 1 flight into the respective orbits.
- The 100% success rate in SSO is more impressive, with 5 successful flights.
- There is little relationship between Flight Number and Success Rate for GTO.
- Generally, as Flight Number increases, the success rate increases. This is most extreme for LEO, where unsuccessful landings only occurred for the low flight numbers (early launches).



Payload vs. Orbit Type

This scatter plot of Orbit Type vs. Payload Mass shows that:

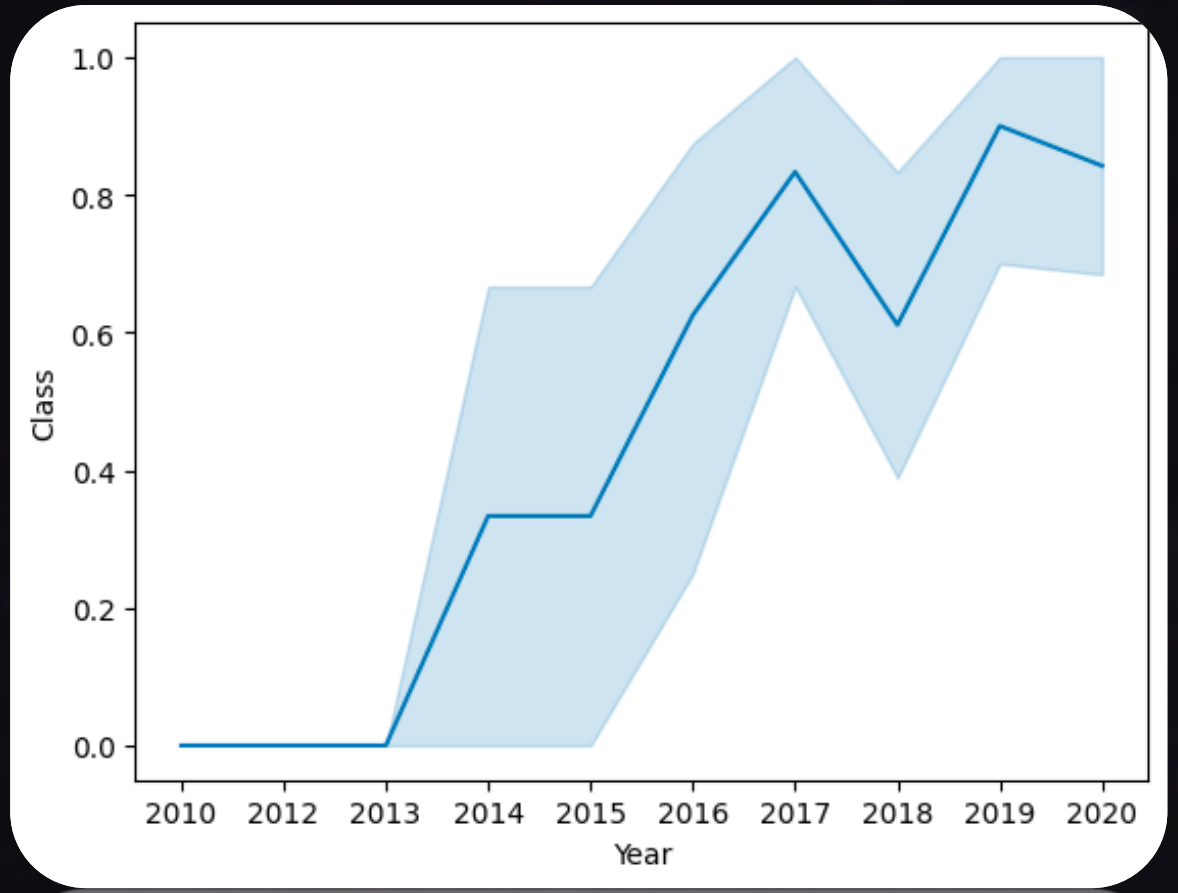
- The following orbit types have more success with heavy payloads:
  - PO (although the number of data points is small)
  - ISS
  - LEO
- For GTO, the relationship between payload mass and success rate is unclear.
- VLEO (Very Low Earth Orbit) launches are associated with heavier payloads, which makes intuitive sense.



Launch Success Yearly Trend

The line chart of yearly average success rate shows that:

- Between 2010 and 2013, all landings were unsuccessful (as the success rate is 0).
- After 2013, the success rate generally increased, despite small dips in 2018 and 2020.
- After 2016, there was always a greater than 50% chance of success.



EDA RESULTS – WITH SQL

All Launch Site Names

Find the names of the unique launch sites

```
%sql SELECT DISTINCT Launch_Site FROM SPACEXTABLE;
```

```
 * sqlite:///my_data1.db
Done.
```

Launch_Site
CCAFS LC-40
VAFB SLC-4E
KSC LC-39A
CCAFS SLC-40

The word DISTINCT returns only unique values from the Launch_Site column of the SPACEXTABLE table.

Launch Site Names Begin with 'CCA'

Find 5 records where launch sites begin with `CCA`

```
%%sql SELECT * FROM SPACEXTABLE
WHERE Launch_Site LIKE 'CCA%' LIMIT 5;
```

Python

```
 * sqlite:///my_data1.db
Done.
```

Date	Time (UTC)	Booster_Version	Launch_Site	Payload	PAYLOAD_MASS__KG_	Orbit	Customer	Mission_Outcome	Landing_Outcome
2010-04-06	18:45:00	F9 v1.0 B0003	CCAFS LC-40	Dragon Spacecraft Qualification Unit	0	LEO	SpaceX	Success	Failure (parachute)
2010-08-12	15:43:00	F9 v1.0 B0004	CCAFS LC-40	Dragon demo flight C1, two CubeSats, barrel of Brouere cheese	0	LEO (ISS)	NASA (COTS) NRO	Success	Failure (parachute)
2012-05-22	07:44:00	F9 v1.0 B0005	CCAFS LC-40	Dragon demo flight C2	525	LEO (ISS)	NASA (COTS)	Success	No attempt
2012-08-10	00:35:00	F9 v1.0 B0006	CCAFS LC-40	SpaceX CRS-1	500	LEO (ISS)	NASA (CRS)	Success	No attempt
2013-01-03	15:10:00	F9 v1.0 B0007	CCAFS LC-40	SpaceX CRS-2	677	LEO (ISS)	NASA (CRS)	Success	No attempt

LIMIT 5 fetches only 5 records, and the LIKE keyword is used with the wild card ‘CCA%’ to retrieve string values beginning with ‘CCA’.

Total Payload Mass

Calculate the total payload carried by boosters from NASA

```
%%sql SELECT SUM(PAYLOAD_MASS__KG_) as TotalPayloadMass FROM SPACEXTABLE
WHERE Customer = 'NASA (CRS)';
```

```
 * sqlite:///my_data1.db
Done.
```

TotalPayloadMass
45596

The SUM keyword is used to calculate the total of the LAUNCH column, and the SUM keyword (and the associated condition) filters the results to only boosters from **NASA (CRS)**.

Average Payload Mass by F9 v1.1

Calculate the average payload mass carried by booster version F9 v1.1

```
%%sql SELECT AVG(PAYLOAD_MASS__KG_) as AvgPayloadMass FROM SPACEXTABLE
WHERE Booster_Version = 'F9 v1.1';
```

```
 * sqlite:///my_data1.db
Done.
```

AvgPayloadMass
2928.4

The AVG keyword is used to calculate the average of the PAYLOAD_MASS__KG_ column, and the WHERE keyword (and the associated condition) filters the results to only the F9 v1.1 booster version.

First Successful Ground Landing Date

Find the dates of the first successful landing outcome on ground pad

```
%%sql SELECT MIN(Date) AS FirstSuccessfulLandingDate FROM SPACEXTABLE
WHERE Landing_Outcome = 'Success (ground pad)';
```

```
 * sqlite:///my_data1.db
Done.
```

FirstSuccessfulLandingDate
2015-12-22

The MIN keyword is used to calculate the minimum of the DATE column, i.e. the first date, and the WHERE keyword (and the associated condition) filters the results to only the successful ground pad landings.

Successful Drone Ship Landing with Payload between 4000 and 6000

List the names of boosters which have successfully landed on drone ship and had payload mass greater than 4000 but less than 6000

```
%sql SELECT Booster_Version FROM SPACEXTBL \
    WHERE (Landing_Outcome = 'Success (drone ship)') AND (PAYLOAD_MASS__KG_ BETWEEN 4000 AND 6000);
```

```
 * sqlite:///my_data1.db
Done.
```

Booster_Version
F9 FT B1022
F9 FT B1026
F9 FT B1021.2
F9 FT B1031.2

The WHERE keyword is used to filter the results to include only those that satisfy both conditions in the brackets (as the AND keyword is also used). The BETWEEN keyword allows for 4000 < x < 6000 values to be selected.

Total Number of Successful and Failure Mission Outcomes

Calculate the total number of successful and failure mission outcomes

```
%%sql SELECT Mission_Outcome, COUNT(*) AS TotalCount
FROM SPACEXTABLE
GROUP BY Mission_Outcome;
```

```
 * sqlite:///my_data1.db
Done.
```

Mission_Outcome	TotalCount
Failure (in flight)	1
Success	98
Success	1
Success (payload status unclear)	1

The COUNT keyword is used to calculate the total number of mission outcomes, and the GROUPBY keyword is also used to group these results by the type of mission outcome.

Boosters Carried Maximum Payload

List the names of the booster which have carried the maximum payload mass

```
%%sql SELECT Booster_Version FROM SPACEXTABLE
WHERE PAYLOAD_MASS__KG_ = (SELECT MAX(PAYLOAD_MASS__KG_) FROM SPACEXTABLE);
```

```
 * sqlite:///my_data1.db
Done.
```

Booster_Version
F9 B5 B1048.4
F9 B5 B1049.4
F9 B5 B1051.3
F9 B5 B1056.4
F9 B5 B1048.5
F9 B5 B1051.4
F9 B5 B1049.5
F9 B5 B1060.2
F9 B5 B1058.3
F9 B5 B1051.6
F9 B5 B1060.3
F9 B5 B1049.7

A subquery is used here. The SELECT statement within the brackets finds the maximum payload, and this value is used in the WHERE condition. (The DISTINCT keyword is then can be used to retrieve only distinct /unique booster versions.)

2015 Launch Records

List the failed landing_outcomes in drone ship, their booster versions, and launch site names for in year 2015

```
%%sql SELECT substr(Date, 6, 2) as Month, Landing_Outcome, Booster_Version, Launch_Site FROM SPACEXTABLE
WHERE substr(Date, 1, 4) = '2015' AND Landing_Outcome = 'Failure (drone ship)';
```

```
 * sqlite:///my_data1.db
Done.
```

Month	Landing_Outcome	Booster_Version	Launch_Site
10	Failure (drone ship)	F9 v1.1 B1012	CCAFS LC-40
04	Failure (drone ship)	F9 v1.1 B1015	CCAFS LC-40

The WHERE keyword is used to filter the results for only failed landing outcomes, AND only for the year of 2015.

Rank Landing Outcomes Between 2010-06-04 and 2017-03-20

Rank the count of landing outcomes (such as Failure (drone ship) or Success (ground pad)) between the date 2010-06-04 and 2017-03-20, in descending order

```
%%sql SELECT Landing_Outcome, COUNT(Landing_Outcome) AS TOTAL_NUMBER FROM SPACEXTABLE
    WHERE DATE BETWEEN '2010-06-04' AND '2017-03-20'
    GROUP BY Landing_Outcome
    ORDER BY TOTAL_NUMBER DESC;
```

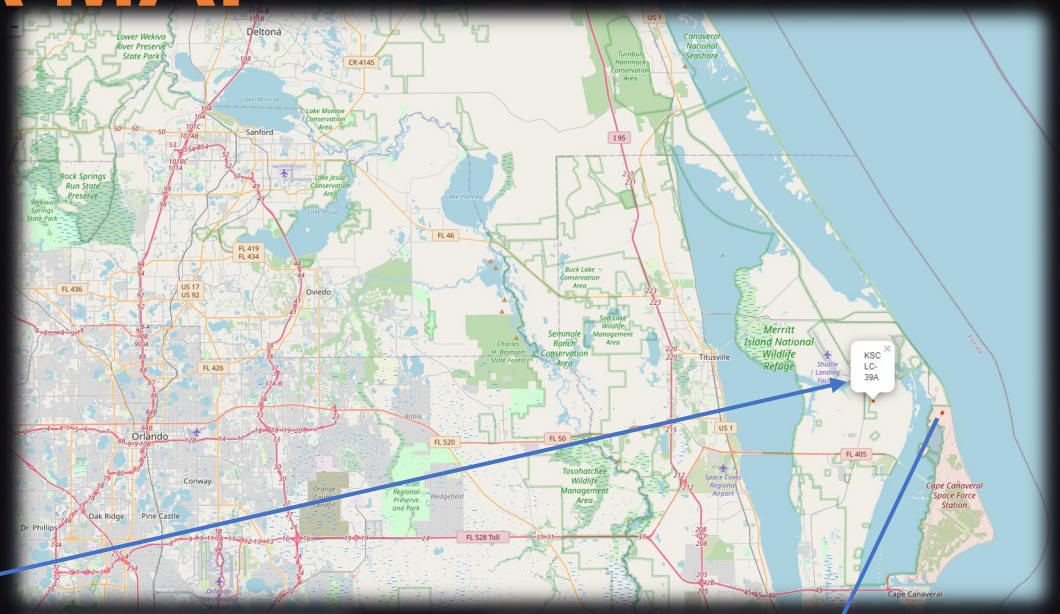
```
 * sqlite:///my_data1.db
Done.
```

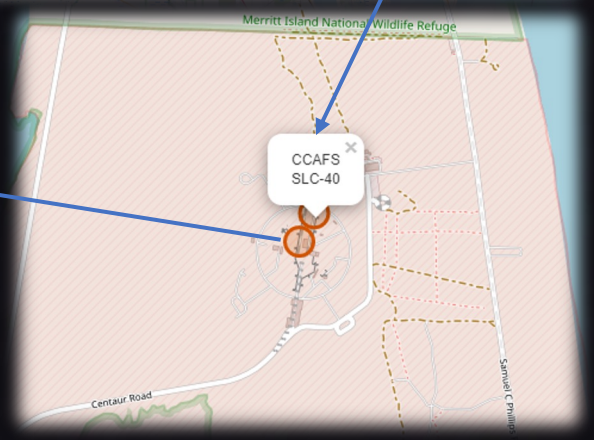
Landing_Outcome	TOTAL_NUMBER
No attempt	10
Success (ground pad)	5
Success (drone ship)	5
Failure (drone ship)	5
Controlled (ocean)	3
Uncontrolled (ocean)	2
Precluded (drone ship)	1
Failure (parachute)	1

The WHERE keyword is used with the BETWEEN keyword to filter the results to dates only within those specified. The results are then grouped and ordered, using the keywords GROUP BY and ORDER BY, respectively, where DESC is used to specify the descending order.

LAUNCH SITES PROXIMITIES ANALYSIS

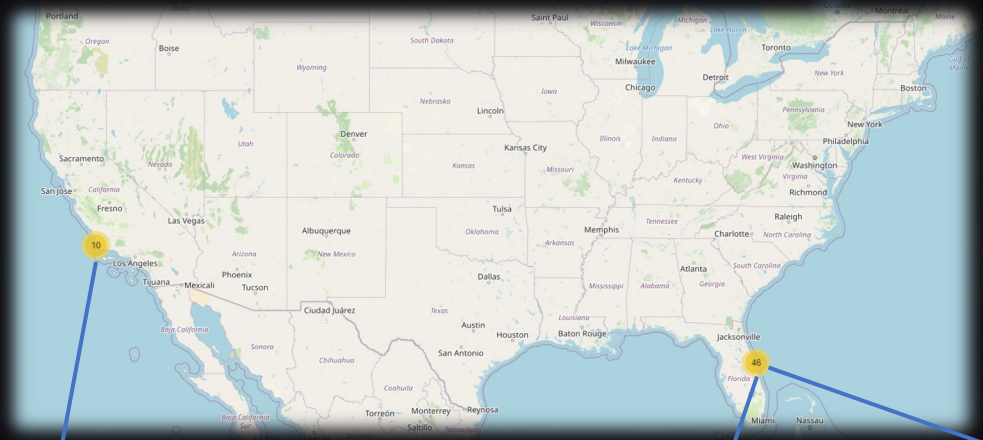
ALL LAUNCH SITES ON A MAP





All SpaceX launch sites are on coasts of the United States of America, specifically Florida and California.

SUCCESS/FAILED LAUNCHES FOR EACH SITE



VAFB SLC-4E

KSC LC-39A

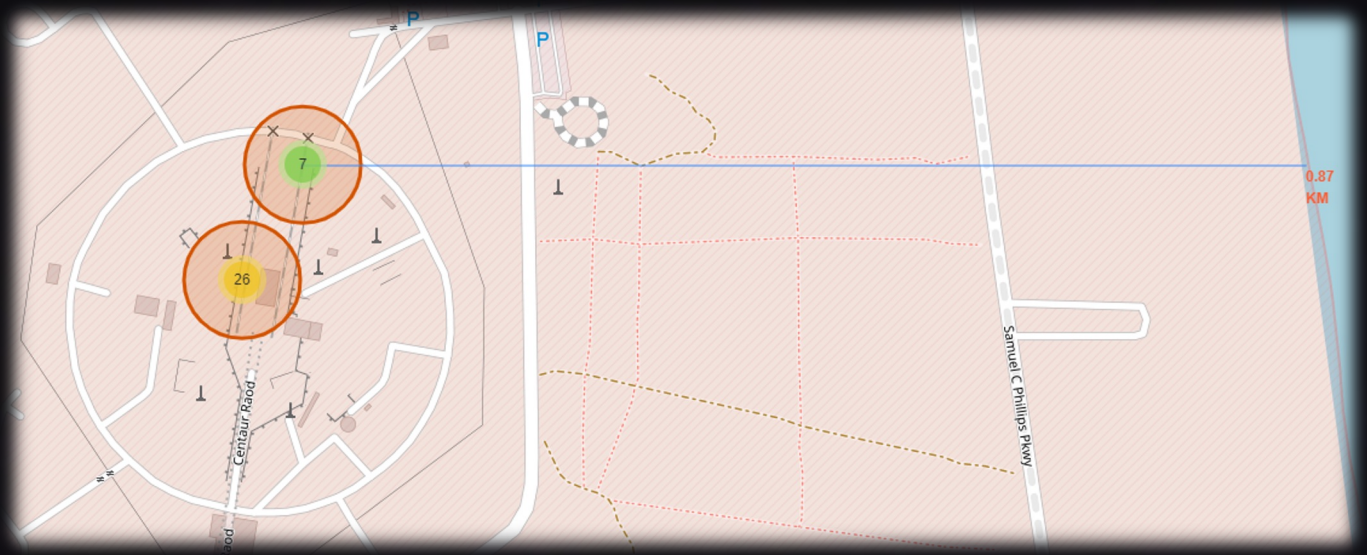
Launches have been grouped into clusters, and annotated with green icons for successful launches, and red icons for failed launches.

CCAFS SLC-40 and CCAFS LC-40

=

PROXIMITY OF LAUNCH SITES TO OTHER POINTS OF INTEREST

Using the CCAFS SLC-40 launch site as an example site, we can understand more about the placement of launch sites.



Are launch sites in close proximity to railways?

- YES. The coastline is only 0.87 km due East.

Are launch sites in close proximity to highways?

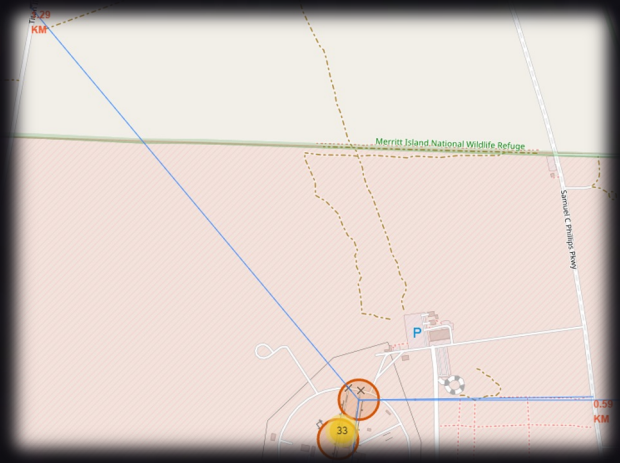
- YES. The nearest highway is only 0.59km away.

Are launch sites in close proximity to railways?

- YES. The nearest railway is only 1.29 km away.

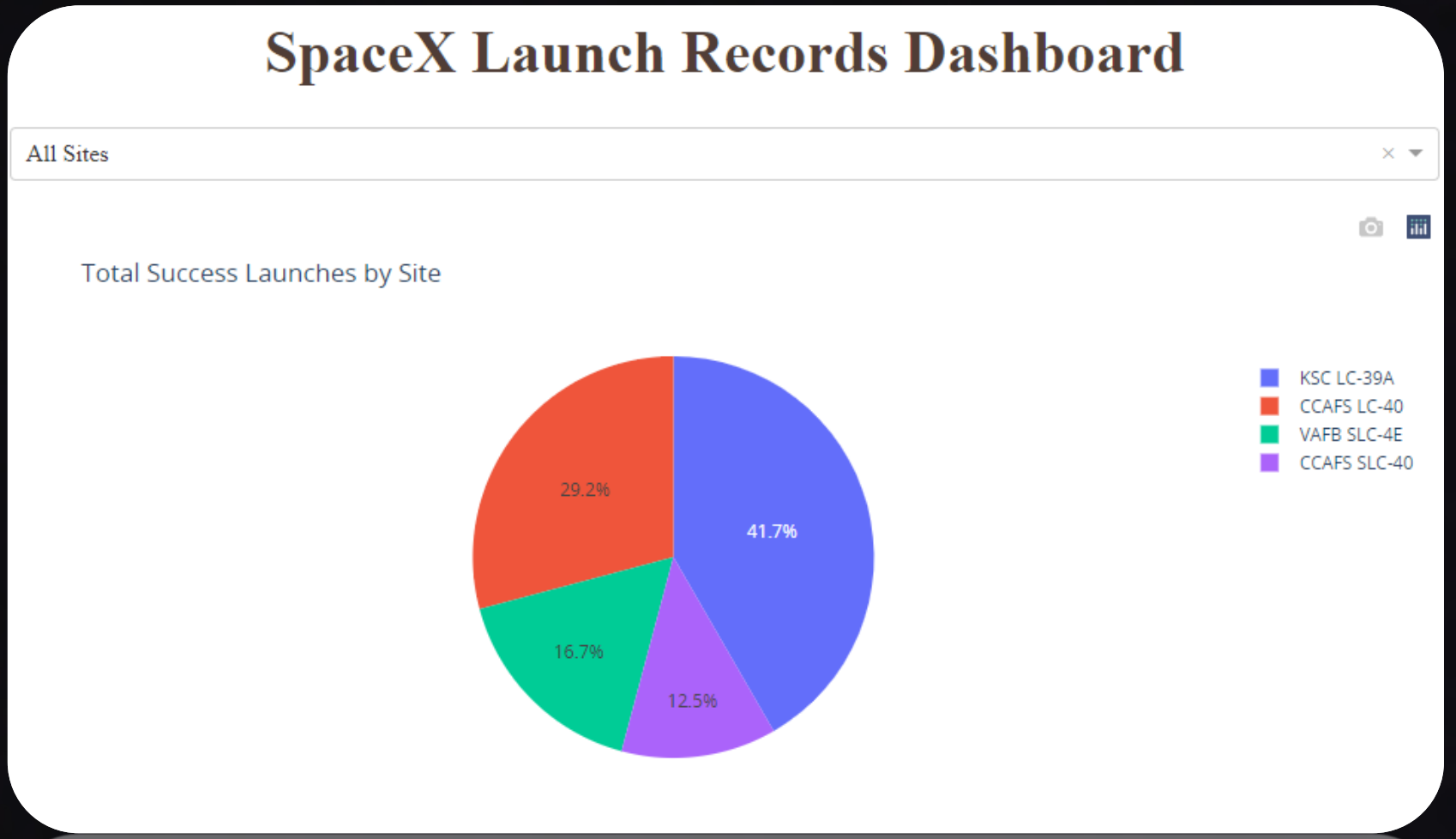
Do launch sites keep certain distance away from cities?

- YES. The nearest city is 51.74 km away.



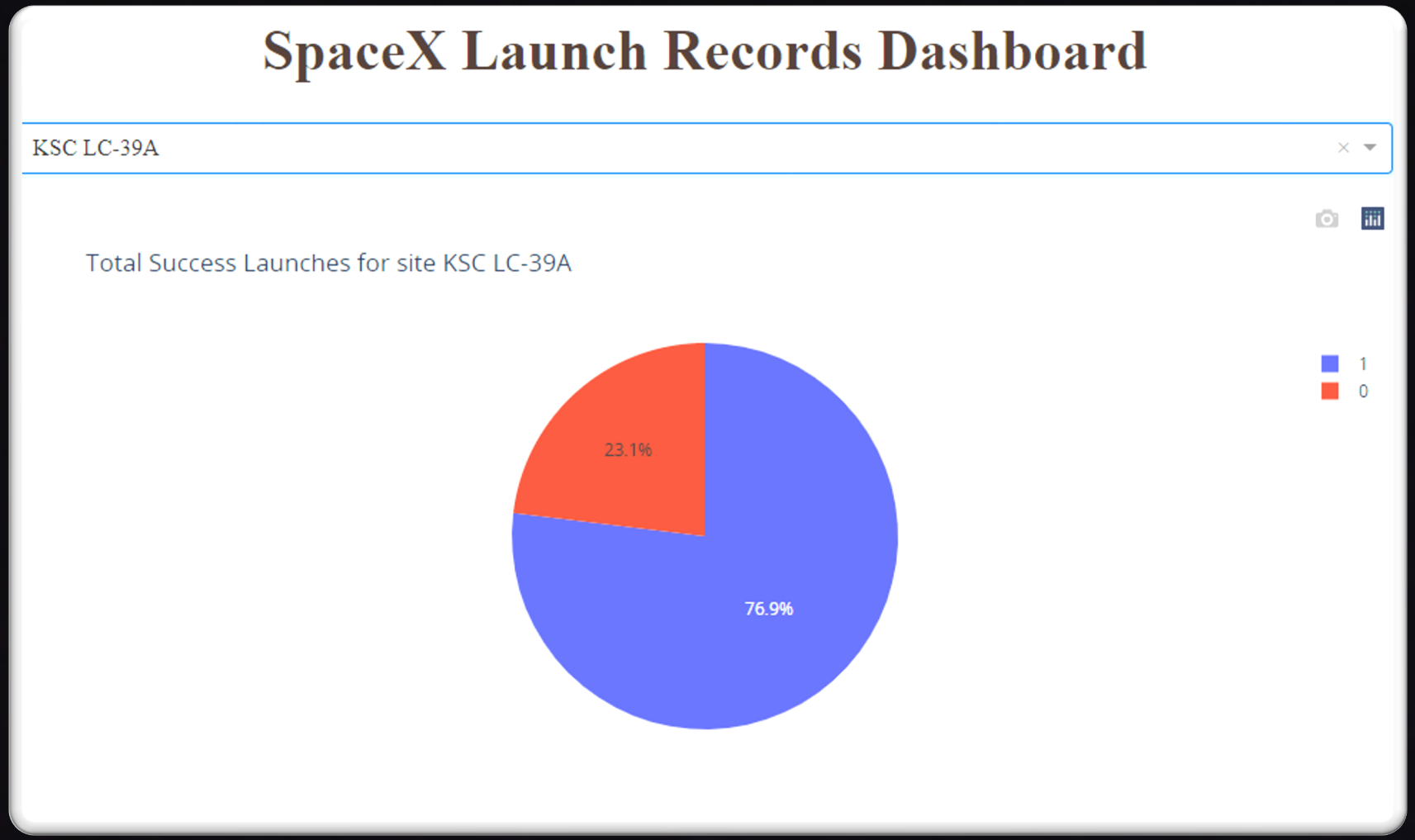
INTERACTIVE DASHBOARD – PLOTLY DASH

Launch Success Count For All Sites



The launch site KSC LC-39 A had the most successful launches, with 41.7% of the total successful launches.

Pie Chart For The Launch Site With Highest Launch Success Ratio



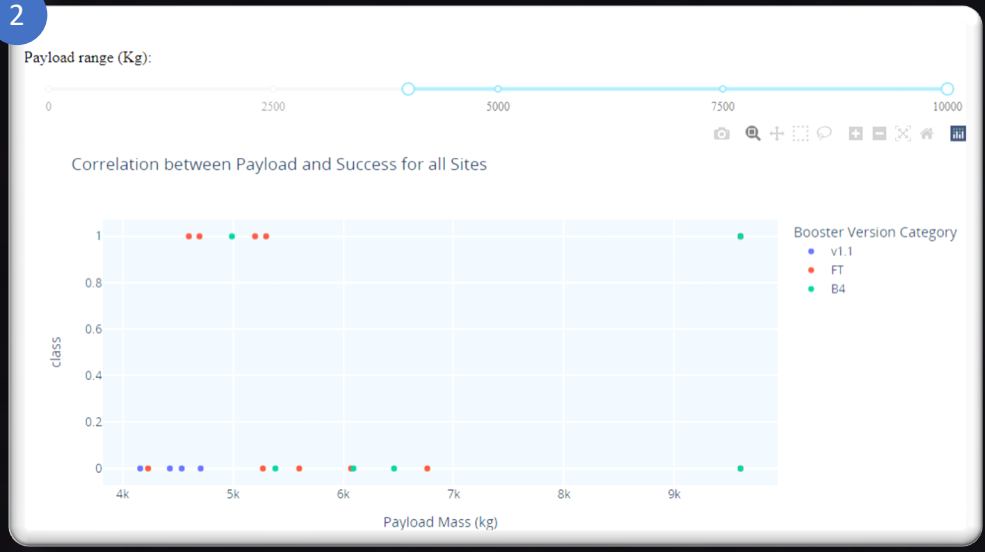
The launch site KSC LC-39 A also had the highest rate of successful launches, with a 76.9% success rate.

Launch Outcome VS. Payload Scatter Plot For All Sites

- Plotting the launch outcome vs. payload for all sites shows a gap around 4000 kg, so it makes sense to split the data into 2 ranges:
  - 0 – 4000 kg (low payloads)
  - 4000 – 10000 kg (massive payloads)
- From these 2 plots, it can be shown that the success for massive payloads is lower than that for low payloads.
- It is also worth noting that some booster types (v1.0 and B5) have not been launched with massive payloads.

1

2



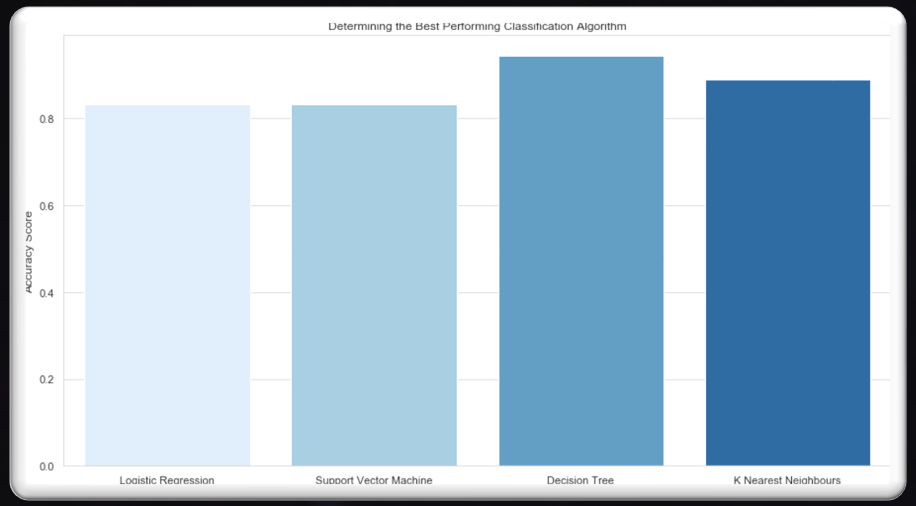
PREDICTIVE ANALYSIS (CLASSIFICATION)

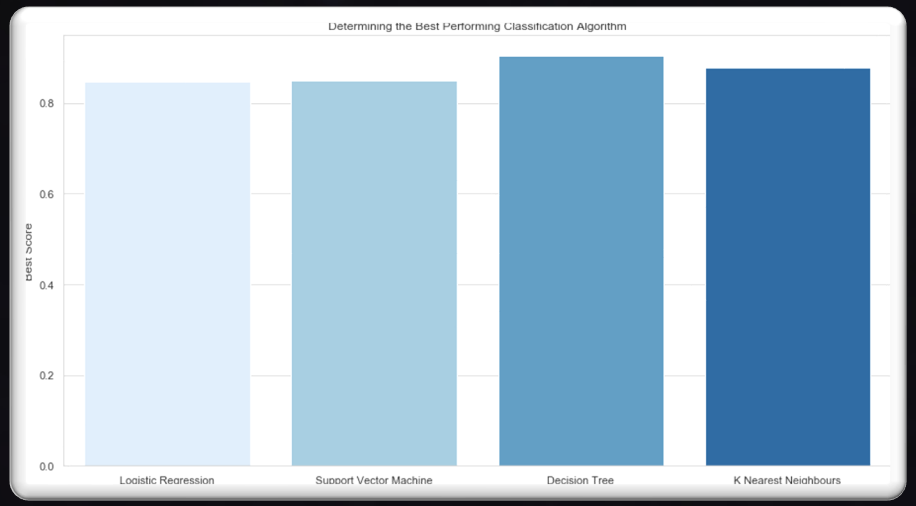
Classification Accuracy

Plotting the Accuracy Score and Best Score for each classification algorithm produces the following result:

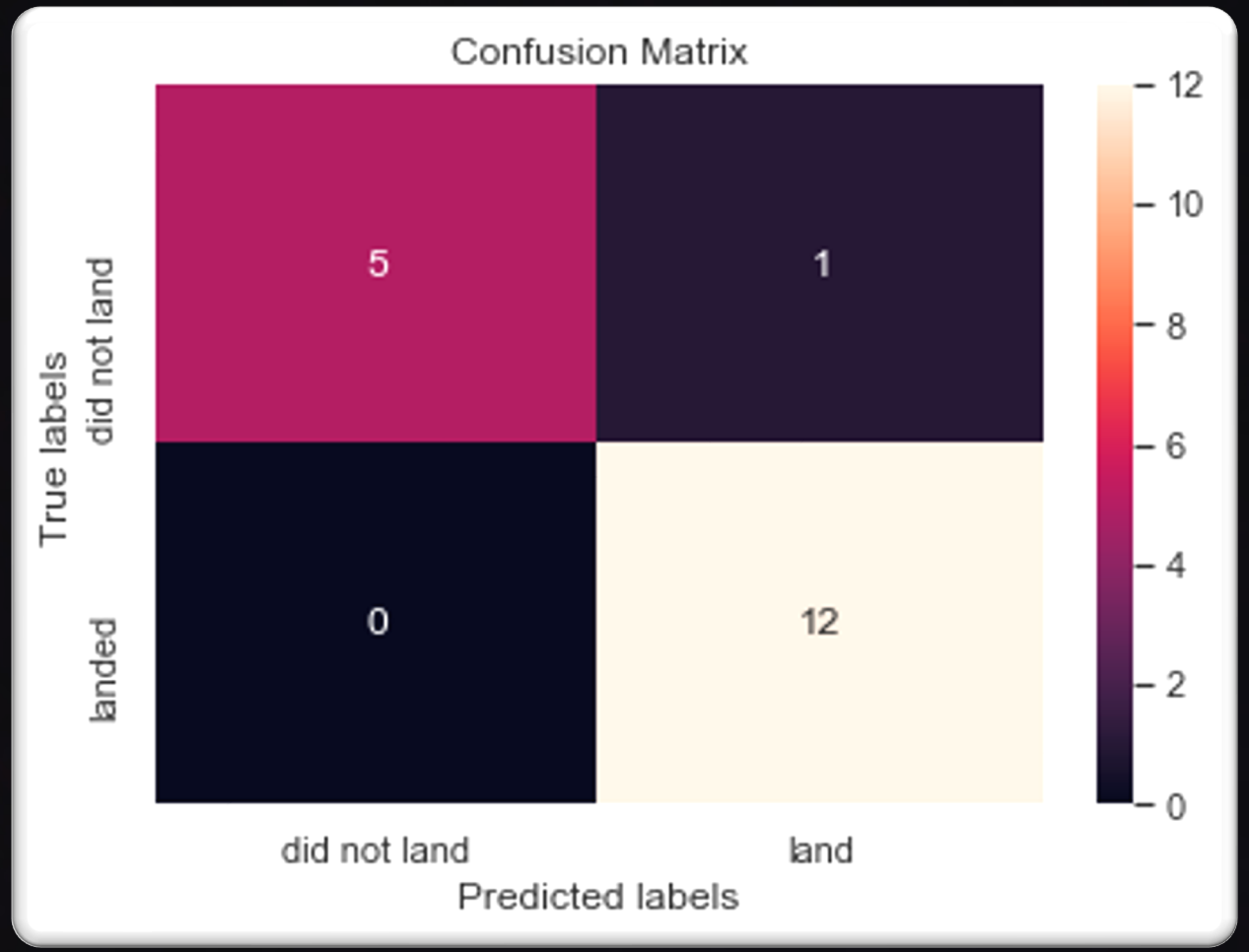
- The Decision Tree model has the highest classification accuracy
  - The Accuracy Score is 94.44%
  - The Best Score is 90.36%

Algorithm	Accuracy Score	Best Score
Logistic Regression	0.833333	0.846429
Support Vector Machine	0.833333	0.848214
Decision Tree	0.944444	0.903571
K Nearest Neighbours	0.888889	0.876786





Confusion Matrix



- The Decision Tree model, with an accuracy of 94.44%, is the best-performing classification model, as was previously demonstrated.
- The confusion matrix, which only displays 1 out of a total of 18 results that were mistakenly categorised (a false positive, shown in the top-right corner), explains this.
- The remaining 17 results (5 didn't land, 12 did) are accurately categorized.

CONCLUSIONS

Conclusions

- Most early flights are unsuccessful, and as the number of flights increases, so does the success rate at a launch location. Consequently, the success rate increases as experience level increases.
  - Between 2010 and 2013, all landings were unsuccessful (as the success rate is 0).
  - After 2013, the success rate generally increased, despite small dips in 2018 and 2020.
  - After 2016, there was always a greater than 50% chance of success.
- Orbit types ES-L1, GEO, HEO, and SSO, have the highest (100%) success rate.
  - The 100% success rate of GEO, HEO, and ES-L1 orbits can be explained by only having 1 flight into the respective orbits.
  - The 100% success rate in SSO is more impressive, with 5 successful flights.
  - The orbit types PO, ISS, and LEO, have more success with heavy payloads:
  - VLEO (Very Low Earth Orbit) launches are associated with heavier payloads, which makes intuitive sense.
- The launch site KSC LC-39 A had the most successful launches, with 41.7% of the total successful launches, and also the highest rate of successful launches, with a 76.9% success rate.
- The success for massive payloads (over 4000kg) is lower than that for low payloads.
- The best performing classification model is the Decision Tree model, with an accuracy of 94.44%.

APPENDIX

DATA COLLECTION – Space X REST API

- Custom functions to retrieve the required information
- Custom logic to clean the data

```python
# Lets take a subset of our dataframe keeping only the features we want and the flight number, and date_utc.
data = data[['rocket', 'payloads', 'launchpad', 'cores', 'flight_number', 'date_utc']]

# We will remove rows with multiple cores because those are falcon rockets with 2 extra rocket boosters
# and rows that have multiple payloads in a single rocket.
data = data[data['cores'].map(len)==1]
data = data[data['payloads'].map(len)==1]

# Since payloads and cores are lists of size 1 we will also extract the single value in the list and replace the feature.
data['cores'] = data['cores'].map(lambda x : x[0])
data['payloads'] = data['payloads'].map(lambda x : x[0])

# We also want to convert the date_utc to a datetime datatype and then extracting the date leaving the time
data['date'] = pd.to_datetime(data['date_utc']).dt.date

# Using the date we will restrict the dates of the launches
data = data[data['date'] <= datetime.date(2020, 11, 13)]
```

Python

From the `rocket` column we would like to learn the booster name.

```python
# Takes the dataset and uses the rocket column to call the API and append the data to the list
def getBoosterVersion(data):
    for x in data['rocket']:
        response = requests.get("https://api.spacexdata.com/v4/rockets/"+str(x)).json()
        BoosterVersion.append(response['name'])
```

Python

From the `launchpad` we would like to know the name of the launch site being used, the longitude, and the latitude.

```python
# Takes the dataset and uses the launchpad column to call the API and append the data to the list
def getLaunchSite(data):
    for x in data['launchpad']:
        response = requests.get("https://api.spacexdata.com/v4/launchpads/"+str(x)).json()
        Longitude.append(response['longitude'])
        Latitude.append(response['latitude'])
        LaunchSite.append(response['name'])
```

Python

From the `payload` we would like to learn the mass of the payload and the orbit that it is going to.

```python
# Takes the dataset and uses the payloads column to call the API and append the data to the lists
def getPayloadData(data):
    for load in data['payloads']:
        response = requests.get("https://api.spacexdata.com/v4/payloads/"+load).json()
        PayloadMass.append(response['mass_kg'])
        Orbit.append(response['orbit'])
```

Python

From `cores` we would like to learn the outcome of the landing, the type of the landing, number of flights with that core, whether gridfins were used, whether the core is reused, whether legs were used, the landing pad used, the block of the core (which is a number used to separate versions of cores), the number of times this specific core has been reused, and the serial of the core.

```python
# Takes the dataset and uses the cores column to call the API and append the data to the lists
def getCoreData(data):
    for core in data['cores']:
        if core['core'] != None:
            response = requests.get("https://api.spacexdata.com/v4/cores/"+core['core']).json()
            Block.append(response['block'])
            ReusedCount.append(response['reuse_count'])
            Serial.append(response['serial'])
        else:
            Block.append(None)
            ReusedCount.append(None)
            Serial.append(None)
        Outcome.append(str(core['landing_success'])+' '+str(core['landing_type']))
        Flights.append(core['flight'])
        GridFins.append(core['gridfins'])
        Reused.append(core['reused'])
        Legs.append(core['legs'])
        LandingPad.append(core['landpad'])
```

Python

DATA COLLECTION – Web Scrapping

- Custom functions for web scraping
- Custom logic to fill up the launch_dict values with values from the launch tables

```
def date_time(table_cells):
    """
    This function returns the data and time from the HTML  table cell
    Input: the  element of a table data cell extracts extra row
    """
    return [data_time.strip() for data_time in list(table_cells.strings)][0:2]

def booster_version(table_cells):
    """
    This function returns the booster version from the HTML  table cell
    Input: the  element of a table data cell extracts extra row
    """
    out=''.join([booster_version for i,booster_version in enumerate( table_cells.strings) if i%2==0][0:-1])
    return out

def landing_status(table_cells):
    """
    This function returns the landing status from the HTML table cell
    Input: the  element of a table data cell extracts extra row
    """
    out=[i for i in table_cells.strings][0]
    return out


def get_mass(table_cells):
    mass=unicodedata.normalize("NFKD", table_cells.text).strip()
    if mass:
        mass.find("kg")
        new_mass=mass[0:mass.find("kg")+2]
    else:
        new_mass=0
    return new_mass


def extract_column_from_header(row):
    """
    This function returns the landing status from the HTML table cell
    Input: the  element of a table data cell extracts extra row
    """
    if (row.br):
        row.br.extract()
    if row.a:
        row.a.extract()
    if row.sup:
        row.sup.extract()

    colunm_name = ' '.join(row.contents)

    # Filter the digit and empty names
    if not(colunm_name.strip().isdigit()):
        colunm_name = colunm_name.strip()
        return colunm_name
```

```
extracted_row = 0
#Extract each table
for table_number,table in enumerate(soup.find_all('table',"wikitable plainrowheaders collapsible")):
   # get table row
    for rows in table.find_all("tr"):
        #check to see if first table heading is as number corresponding to launch a number
        if rows.th:
            if rows.th.string:
                flight_number=rows.th.string.strip()
                flag=flight_number.isdigit()
        else:
            flag=False
        #get table element
        row=rows.find_all('td')
        #if it is number save cells in a dictonary
        if flag:
            extracted_row += 1
            # Flight Number value
            # Append the flight_number into launch_dict with key `Flight No.`
            launch_dict["Flight No."].append(flight_number)

            # Date value
            #Append the date into launch_dict with key `Date`
            datatimelist=date_time(row[0])
            date = datatimelist[0].strip(',')
            launch_dict["Date"].append(date)

            # Time value
            #Append the time into launch_dict with key `Time`
            time = datatimelist[1]
            launch_dict["Time"].append(time)

            # Booster version
            #Append the bv into launch_dict with key `Version Booster`
            bv=booster_version(row[1])
            if not(bv):
                bv=row[1].a.string
            launch_dict["Version Booster"].append(bv)

            # Launch Site
            #Append the bv into launch_dict with key `Launch site`
            launch_site = row[2].a.string
            launch_dict['Launch site'].append(launch_site)
```

```
            # Payload
            #Append the payload into launch_dict with key `Payload`
            payload = row[3].a.string
            launch_dict['Payload'].append(payload)

            # Payload Mass
            #Append the payload_mass into launch_dict with key `Payload mass`
            payload_mass = get_mass(row[4])
            launch_dict['Payload mass'].append(payload_mass)

            # Orbit
            #Append the orbit into launch_dict with key `Orbit`
            orbit = row[5].a.string
            launch_dict['Orbit'].append(orbit)

            # Customer
            #Append the customer into launch_dict with key `Customer`
            if row[6].a != None:
                customer = row[6].a.string
            else:
                customer = 'None'

            launch_dict['Customer'].append(customer)

            # Launch outcome
            #Append the launch_outcome into launch_dict with key `Launch outcome`
            launch_outcome = list(row[7].strings)[0]
            launch_dict['Launch outcome'].append(launch_outcome)

            # Booster Landing
            #Append the booster landing into launch_dict with key `Booster Landing`
            booster_landing = landing_status(row[8])
            launch_dict['Booster landing'].append(booster_landing)


            print(f"Flight Number: {flight_number}, Date: {date}, Time: {time} \n \
            Booster Version {bv}, Launch Site: {launch_site} \n \
            Payload: {payload}, Orbit: {orbit} \n \
            Customer: {customer}, Launch Outcome: {launch_outcome}\
            Booster Landing: {booster_landing} \n \
            *** ")
```